Exhibit F

Approved for public release SAND2018-5190C

Convolutional Back Projection on the S1 Reduced Precision Processor

June 25-29, 2018

Michael Holzrichter and Randy Spaulding

Sandia National Laboratories

P.O. Box 5800 MS 0501, Albuquerque, NM 87185-0501

Abstract

Sandia's synthetic aperture radar (SAR) business area investigated reduced precision computing as a way to increase the performance of processing raw radar data in a way that does not rely on Moore's Law. The concept is that if less precision can be tolerated, then more parallelism can be obtained from the same number of transistors. This means increased processing performance without increasing size, weight and power (SWaP).

A core effort was an evaluation of the S1 processor from Singular Computing. The S1 is a first-generation implementation of Singular Computing's approach to reduced precision computation. The S1 has a large number of simple computing elements (called "approximate processing elements" or APEs) that carry out reduced precision computations. Each S1 chip has 2112 approximate processing elements, each of which has 512 2-byte words of memory. The S1 implements SIMD parallelism with all processing elements executing the same instruction stream in lock-step.

Sandia implemented the compute-intensive convolutional back projection (CBP) [1], [2] image formation algorithm on the S1. This entailed overcoming challenges associated with the primitive capabilities and limited memory of the S1's APEs. Systems built around the S1 have either one or sixteen S1 chips. The memory requirements to process a 1024 x 1024-pixel SAR image required a 16-chip system.

Sandia evaluated the performance of its S1 implementation of the CBP algorithm. Sandia produced good quality SAR images using reduced precision computations, thus demonstrating the utility of reduced precision computation to mitigate the expected demise of Moore's Law. However, Sandia found that its implementation of CBP on the S1 is much slower than its implementation of the same algorithm on the Tegra X1 GPU.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia found that the S1's ratio of compute, memory and I/O capabilities was not well-matched to the needs of the CBP algorithm. This mismatch of resources impacted performance. Singular Computing reviewed Sandia's S1 code and suggested alternative approaches that it felt had the potential to speed up the code by an order of magnitude or more. However, this would not be enough to overcome the greater than two orders of magnitude speed advantage of the GPU implementation. A core problem is the S1's slow I/O is not amortized over enough computation. If the CBP results fed directly into another compute-intensive algorithm, such as an on-chip deep neural net, then the I/O would be amortized over much more compute and the conclusions we describe here might change.

Keywords

Reduced precision computation, approximate computing, convolutional back projection, synthetic aperture radar (SAR) image formation, S1 processor, Moore's Law

Introduction

With the expectation that Moore's Law cannot drive improvements in processor performance forever, Sandia's synthetic aperture radar (SAR) business area looked into ways to increase performance from processors that do not rely on Moore's Law. One of the strategies considered was reduced precision computing. The concept is that if less precision can be tolerated, then more parallelism can be obtained from the same number of transistors. This means more performance without increasing size, weight and power (SWaP).

In 2013 Sandia became aware of a company called Singular Computing and their DARPA-funded S1. The S1 is a prototype chip that implements reduced precision computing. It has a large number of simple computing elements. The S1 provided Sandia the opportunity to explore reduced precision SAR image formation in a concrete manner. This document summarizes Sandia's initial experience in using the S1 to form SAR images.

This document begins with a summary of the S1 architecture followed by a summary of the convolutional back projection (CBP) image formation algorithm. Following this is a discussion of the implementation of CBP on the S1 including several adaptations to the CBP algorithm that were necessary due to the S1's unique design. This document concludes with a discussion of the performance results and the conclusions drawn from them.

S1 Concept and Architecture

The S1 is a chip from Singular Computing. The S1's primary target is the neural-net application space. Its design is inspired by the brain, in that it strives to maximize parallelism and communication and minimize power consumption [3]. To achieve these goals, the S1 sacrifices precision in computations. In

practice, the S1 increases the amount parallelism obtained from a given number of transistors by limiting the capabilities of the individual processing elements (cores).

At the heart of the S1 architecture are the Approximate Processing Elements (APEs) [4], [5]. Each S1 chip has 2112 APEs. An APE roughly corresponds to a CPU "core", but with restricted capabilities. Each APE has 512 2-byte words of memory for a total of approximately 2 megabytes of memory per chip. Each APE has 8 2-byte general-purpose registers. The S1 has three data types: boolean, 16-bit integers and a 14-bit "approximate float" (occupying a 2-byte word) which is similar to an IEEE half-precision floating point. (For brevity, the term "approx" will be used for "approximate float" in the remainder of this document.)

The operations available for the approx datatype are Add, Sub, Mult, Div and Sqrt. The operations available for integers are Add, Sub, And, Or, ShiftLeft, and ShiftRight. The S1 does not provide machine instructions to multiply or divide integers. General-purpose integer multiply or divide operations must be synthesized from long sequences of existing operations and hence are much slower. All arithmetic machine instructions complete in a single clock cycle.

Within the S1 chip the APEs are arranged in a grid with 48 rows and 44 columns. The APEs operate in a SIMD (Single Instruction, Multiple Data) fashion with all APEs executing in lock-step the same instruction at the same time. The APEs communicate with each other over a NEWS (north, east, south, west) interconnection network.

An S1 *system* consists of a Xilinx Zynq (with ARM processor) running Linux, an FPGA control unit (CU) and 1 or 16 S1 chips. In the 16-chip configuration the S1 chips are logically arranged in a 4 x 4 grid. The Zynq's ARM processor is the host processor with ultimate control over the S1 system and the CU+S1 chips appear to applications running on the ARM as a compute accelerator.

The CU is the source of the control signals for all S1 chips. The CU issues the instruction stream executed by the APEs as well as serves as the gateway via which data enter or leave the S1 processors. The CU has 64K 2-byte words of data memory and 64K 4-byte words of memory for the SIMD instruction stream. In addition to APE-to-APE communication, the CU can broadcast to all APEs simultaneously with all APEs receiving the same value at the same time. While all APEs receive the same value at the same time, each APE decides whether or not to ignore the value based on a local flag. Because fan-out is much easier than fan-in, extracting data out of the grid is different. There are two options. Either an individual register of an individual APE is read or the OR of a register of all APEs can be read.

Programming Model

The programming model of the S1 was inspired by OpenCL. Kernels are constructed programmatically and then executed one or more times. The S1 has two programming environments. The original environment is very low-level in which everything is expressed at the individual operation level, much like assembler language programming. The new "Nova" programming model is not as low-level. Nova is a simple procedural language, with expressions and statements. There is a runtime compiler that users invoke to convert Nova programs into machine code that the accelerator can run.

The Host, CU and each APE have disjoint memory spaces. The programmer must explicitly transfer data between memory spaces. Data can be transferred between (1) Host and CU (2) CU and APEs, and (3)

between neighboring APEs. There is no direct transfer between Host and APEs. The programmer is responsible for the layout and management of data in the CU and APE memory.

One of the central features of the S1 architecture is the approx data type. It is similar to, but not the same as, 14-bit floating point. One of the 14 bits is a sign bit, seven bits are an exponent and the remaining 6 are a "logtissa" i.e. the log of the mantissa. The inherent granularity of the approx data type is 1 part in 64 (about 1%). The approx data type has a wide dynamic range (on the order of $10^{+/-20}$). One of the peculiarities of the approx data type is that it cannot exactly represent zero.

Convolutional Back Projection

FFT-based SAR image formation algorithms, such as polar format, are popular due to their computational efficiency. In broad strokes, the polar format algorithm can be thought of as a specialized 2D discrete Fourier transform. In convolutional back projection (CBP) the 2D DFT is an implicit operation. Because the transform is implicit, it does not get the $O(n^2)$ to $O(n \log n)$ improvement in computational efficiency provided by FFT-based methods but in doing so it relaxes the restriction of uniform sample spacing in Doppler space. Convolutional back projection is considered to be a computationally expensive image formation algorithm but one that avoids approximations made by FFT-based algorithms.

In terms of calculations, CBP builds up the image from phase histories one at a time. Each phase history will make a contribution to each pixel in the image. Each pixel corresponds to a specific location in the phase history. The value of the phase history at that location is phase shifted and added to the running sum for that pixel. The location in the phase history typically lands between phase history samples. Therefore, the phase history is up-sampled enough such that linear interpolation of two adjacent samples provides a good enough approximation.

The location in the phase history that contributes to any given pixel as well as the phase shift to apply prior to summation depends on imaging geometry, including both the pixel's and radar's location relative to the image's scene reference point.

SAR is a coherent process and therefore it depends on getting phase correct. Hence wavelength is a natural unit of length. The geometry involved in the CBP calculations involves distances in the range of hundreds to hundreds of thousands of wavelengths. Coherence demands require these distances be correct to a fraction of a wavelength. This is a requirement that drove fundamental decisions on how to implement CBP on the S1.

Convolutional Back Projection S1 Implementation

This section discusses the mapping of the CBP algorithm onto the S1 architecture. There are two basic approaches to mapping the CBP algorithm onto a parallel architecture. They differ in the methodology whereby pixels and the phase history samples that are summed to form them come to be collocated at the processor which performs the summation. The distinction between the two approaches is whether it is pixels or phase history samples that are assigned to a fixed processor.

(a) In the first method, phase history samples are partitioned across processors. Once a phase history sample is in the memory of its assigned processor it stays there. The parallel communication task is to get the running sum for each pixel to the processors holding the phase history samples that contribute to it.
(b) In the second method, it is the pixels that are partitioned across processors. A phase history sample may contribute to multiple pixels. The parallel communication task is to get each phase history sample to the multiple processors holding the pixels that it contributes to.

In order to minimize communication, the latter strategy was chosen. This meant that the largest image that can be formed is dictated by the amount of memory available to hold the pixels. Each APE's 512-word memory is able to hold 32 pixels along with the associated memory overhead. To a decent approximation, one S1 chip can form 256 x 256 pixels. To form a 1024 x 1024 image requires a system with 16 S1 chips.

In order to build up the pixels, two pieces of information are needed for each phase history-pixel combination:

(1) the index of the phase history sample that maps to the pixel and
(2) the phase shift to apply prior to summation with the pixel's running sum.

These two quantities are very smooth functions of pixel position. This CBP implementation takes advantage of this smoothness by partitioning the image into 16 x 16 pixel tiles and obtaining these quantities inside the tile by interpolating the value of these quantities at the corners of the tiles. Since there are 32 pixels per APE, an APE forms the pixels for two rows of a tile. The pixels for a tile are distributed across 8 APEs.

The choices discussed above for mapping the CBP algorithm onto the S1 architecture dictate the following high-level structure of the algorithm. At the top level, the algorithm has three main steps:

(1) Initialize host and APEs based on problem parameters (size of image, etc.).
(2) Process the phase histories to build up the SAR image in the APE memory.
(3) Extract the SAR image from APE memory.

During initialization (step 1),

- host memory is allocated,
- a layout of APE memory is created,
- kernels are created,
- APEs are initialized.

Initialization takes place once per image. It takes a small percentage of the overall time. Similarly, the extraction of a completed image also represents less than 1% of the total S1 execution time. While extracting data from the S1 is an inherently serial process, image extraction is only done once per image.

The bulk of the work to form an image is related to processing phase histories. The processing of phase histories is implemented as the following six steps. These six steps are repeated for every phase history.

(1) Evaluate sample mapping and phase at control points (host)
(2) Compute interpolation parameters (host)
(3) Distribute interpolation parameters to the APEs (I/O)
(4) Perform interpolation (APE)
(5) Distribute PH data (I/O)
(6) Update image (APE)

The first two steps take place on the host. In these steps, the mapping and phase are evaluated at the control points (located at the corners of the tiles) from which interpolation parameters are derived for the tiles. The interpolation parameters are then distributed to the APEs. This is an I/O and quasi-serial. The interpolation parameters are broadcast to all APEs one set at a time but 8 APEs will record any given set of interpolation parameters.

At the end of the third step all APEs have the interpolation parameters for the tile in which its pixels reside. In the fourth step the interpolation parameters are used to compute the sample numbers that map to the APE's pixels as well as the phase shift to apply prior to summing. Step 4 concludes with each APE sorting its list of sample numbers. Doing so speeds up the next step.

Step 5 is another I/O step. The actual phase history samples are broadcast to the APEs. Each APE listens and records the phase history samples that contribute to its pixels. Since the phase history samples are broadcast in order and the list of sample numbers an APE is looking for was sorted at the end of step 4, an APE always knows the next phase history sample number needed by any of its pixels. This enables each APE to know immediately whether to record or discard the next sample that is broadcast.

At the end of step 5 each APE has the phase history samples and the phase shifts needed to update the pixels it contains. Updating the pixels with the phase history samples and phase shifts takes place in step 6. In step 6, the phase shift is applied to the phase history sample contributing to each pixel and the result is added to each pixel's running sum.

Tailoring the Algorithm to the S1

The architecture of the S1 differs in numerous ways from that of traditional CPUs. Therefore, the optimal implementation of CBP on the S1 differs in many ways from the best implementation on a Pentium. Below are some of the unique aspects of our S1 implementation of CBP.

Limited representation power of approx and 2-byte integers: The approx data type is good to 1 part in 64. However, CBP needs to compute phase to a fraction of a cycle at distances of thousands of wavelengths. Our implementation mitigates this problem via three unique strategies. First, the phase and mapping are computed to full precision on the host at points on a sparse grid. These values are interpolated on the S1. Second, within the S1, these values are represented using a multi-precision, fixed point format using two integers. This format has a uniform representation error which helped suppress error growth as the interpolation calculations march forward. Third, Kahan summation [6] was used for calculation of the actual pixel values. At the expense of double the space and four times the computations, Kahan summation greatly reduces the error of summations using floating-point-type formats.

Memory: The S1 has very limited memory compared to CPU- and GPU-based architectures. For every pixel, there are numerous overhead values that need to be computed alongside the actual pixel value. In order to maximize the number of pixels per APE, the S1 implementation time-division multiplexes the memory used to store these overhead values. In historical contexts, this practice went by the name of "memory overlay".

Irregular I/O pattern: It is difficult to determine *a priori* in the host, an exploitable pattern of which phase history samples go to which APEs. Furthermore, the I/O process of distributing phase history data is essentially serial in the context of the CU; the CU can only provide one sample to the grid at a time.

Because of this, the strategy used by the S1 implementation is to broadcast the phase history data to all APEs simultaneously. The APEs listen for the samples needed by the pixels allotted to them. The process of optimizing the determination of whether a phase history sample is needed by one of the APE's pixels is to compute and sort the sample numbers prior to broadcasting phase histories. Once the sample numbers are sorted the APEs always know the next phase history sample that is needed with O(1) work per sample broadcast.

I/O is a major time consumer: For the S1 implementation of CBP, I/O is out of balance with computation. Much more time is spent getting data in and out of the grid than the computations in the grid. To address this, the S1 CBP implementation strives to minimize I/O. As mentioned previously, this is the reason for the parallelization strategy of a static mapping of pixels to APEs.

Another I/O minimization strategy is to avoid sending phase history samples into the grid that will not be used. The global min and max index of the phase history samples which contributes to any pixel is determined when calculating the interpolation parameters. The actual phase history samples broadcast to the APEs is limited to this range. Samples outside of this range are not broadcast because it is guaranteed no APE will use them.

Primitive programming environment: There is no standard library of common functions. Most notably for CBP there are no standard implementation of sin/cos and IntToApprox(). We wrote our own implementations of these functions. We wrote code that computes sin and cos for a given phase. The argument is an integer between 0 and 1023. A value of N means the argument is N * 2 * pi / 1024 radians. The function uses a look up table with 32 complex entries in approx format. Using these entries as input, a handful of operations calculate the sin/cos values for the argument. We also wrote code that manually converts integers to approx by summing the contribution of blocks of bits at a time.

Missing integer operations: The APEs do not have instructions for multiplying or dividing integers. To overcome this, the code pre-structures the computations so that integer multiplies and divides are by powers of two and can be accomplished in a single operation by bit shifts.

Evaluation of Results

Our S1 implementation of the CBP algorithm discussed in the previous section produced the SAR image in Figure 1. This is an image of Sandia's antenna test range on the south side of the Kirtland Air Force Base. The vertical dark stripe in the center is a slab of smooth concrete with very low backscatter. The grid of bright dots is an array of corner reflectors used for calibrating SAR images. The image in Figure 2 is a reference image created from the same data by a CPU version of CBP using an Intel Pentium processor and 64-bit IEEE floating point. Qualitatively the image produced by the S1 (left image) has reasonable appearance. To the human eye it looks identical to the reference image (right) produced using full precision floating point.

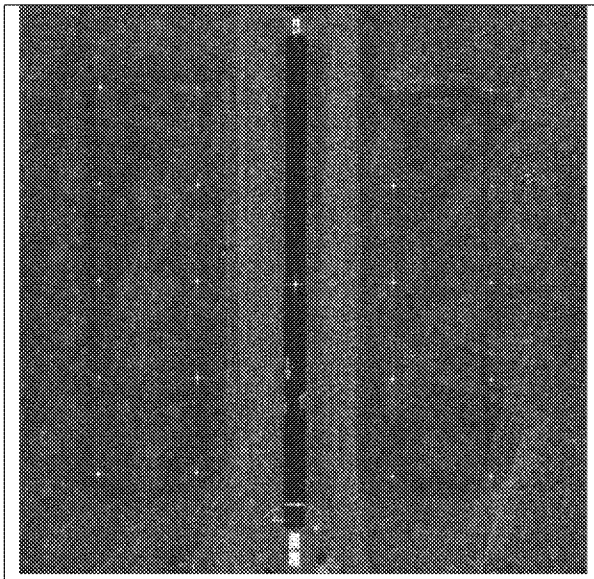

Figure 1 SAR Image Produced by S1

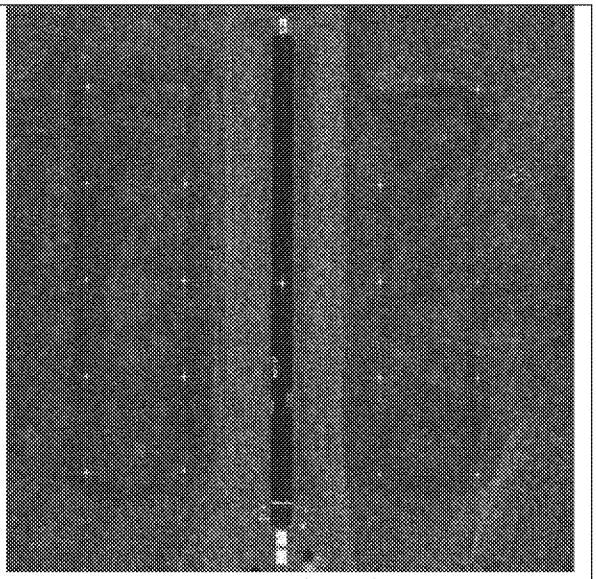

Figure 2 SAR Image Produced by Intel CPU

Difference Image

The image in Figure 3 is the magnitude of the difference between the images in Figure 1 (S1) and Figure 2 (CPU). The differences are small, rarely exceeding 1 dBsm. In order to make the content of the difference image more visible, the brightness of Figure 3 was greatly increased such that a pixel in Figure 3 has the same brightness as a pixel in Figures 1 and 2 with 100 times the magnitude.

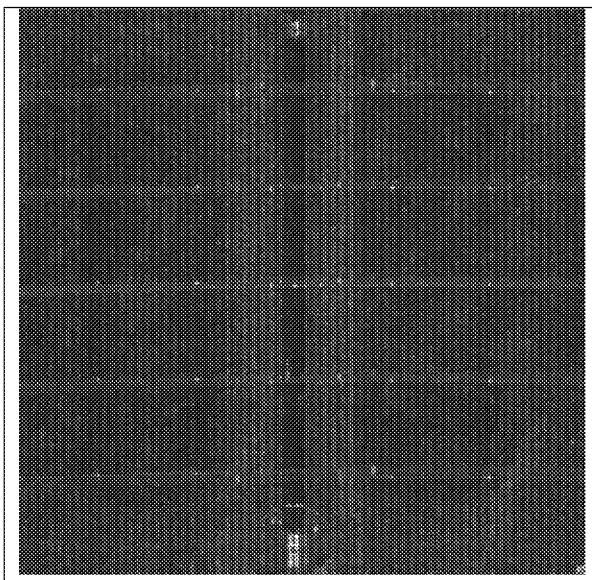

Figure 3 Difference between S1 and CPU images (LUT=15). Difference dominated by corduroy pattern due to errors interpolating phase by bilinear interpolator.

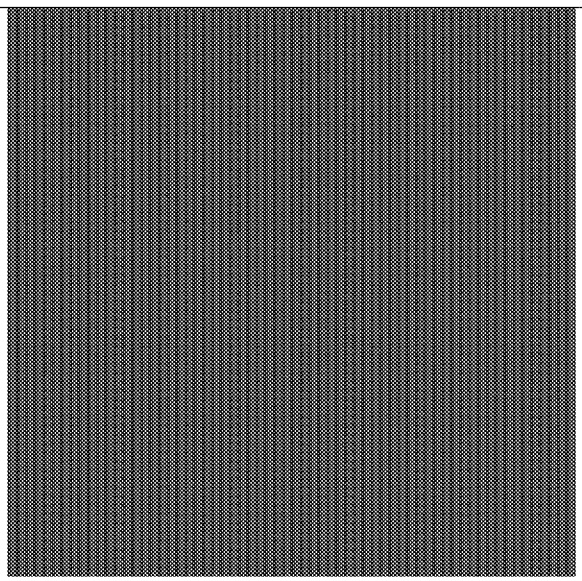

Figure 4 Cumulative (over all phase histories) error in phase introduced by bilinear interpolator.

The difference image is dominated by a vertical corduroy pattern. The pattern matches columns of tiles. The cause of the corduroy pattern is interpolation error of the phase introduced by the bilinear interpolator used to interpolate phase from the corners of the tiles to their interior pixels. Figure 4 shows the interpolation error of phase. The phase function has considerably more curvature in azimuth than range. Because of the small curvature in range, linear interpolation in range introduces minor error. However, the much greater curvature in azimuth means linear interpolation in azimuth introduces much greater error for points in the region half-way between the left and right sides of tiles. In this region, the error of the interpolated phase is on the order of 0.15 radians (9 degrees).

The corduroy error is due to the interpolation method rather than the arithmetic within the S1. Using higher fidelity interpolation in azimuth would considerably reduce the corduroy pattern. The S1 code has been reviewed and the additional resources for a quadratic or cubic interpolator in azimuth are projected to incur minor marginal cost in terms of space or computation.

The second most prominent feature of the difference image are the horizontal streaks emanating from the grid of corner reflectors in the image. This is multiplicative noise from the sidelobes of the response from the corner reflectors. We attribute the horizontal streaks to the reduced precision of the calculations performed in the S1. The corner reflectors produce strong returns at these ranges. The reduced precision calculations do not permit constructive and destructive interference to work as effectively. It is interesting to note there are no horizontal streaks associated with the bright areas at the ends of the concrete area. This is because the RCS of these areas is roughly 10-15 dB lower than that of the corner reflectors so the sidelobes generated are below the level of the surrounding clutter.

Impulse Response

To better quantify the quality of the images, Figures 5 and 6 show the analysis of the impulse response (IPR) of the center reflector in the S1 and CPU images respectively. Overall the IPRs of the two images are similar. The peak in the S1 image is a fraction of a dB weaker than in the CPU image. The differences in the range IPRs are very minor. The differences in the azimuth IPRs are more pronounced.

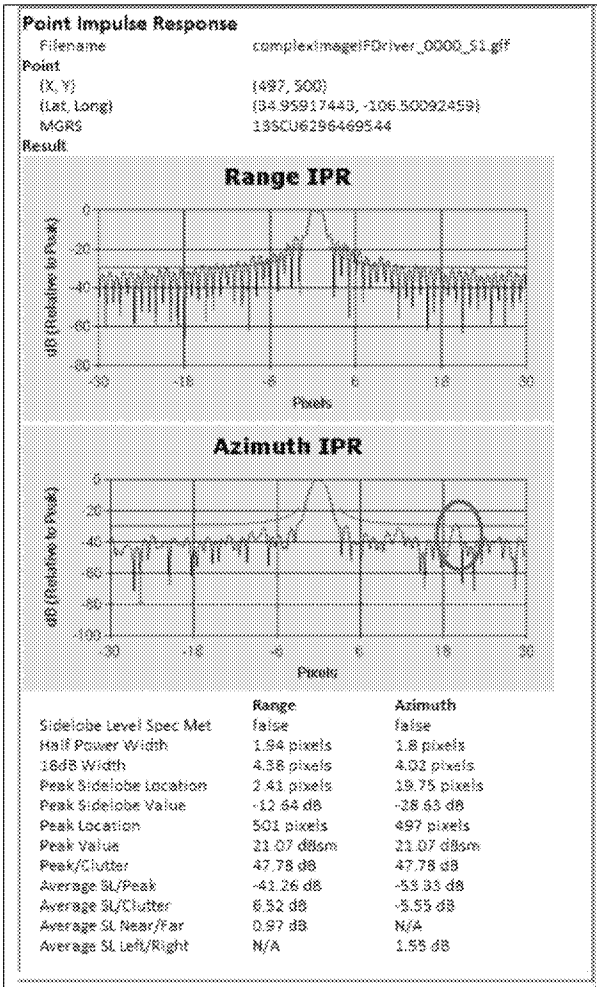

Figure 5 IPR of Center Reflector of S1 Image

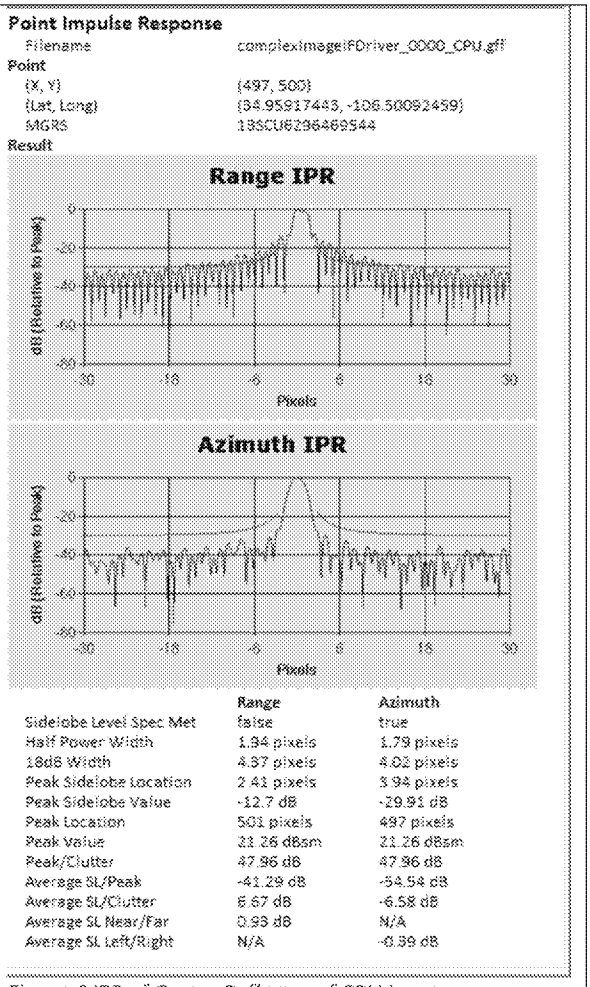

Figure 6 IPR of Center Reflector of CPU Image

The most salient difference in the azimuth IPRs is the distinct sidelobe on the right side of the azimuth IPR for the S1 image that is not present in the CPU azimuth IPR. Beyond this sidelobe, the S1 image's azimuth IPR does not have as deep nulls as the CPU image and the main lobe is slightly broader.

These effects are not enough to substantially change the quantitative metrics appearing at the bottom of the figures. Other than the sidelobe at +20 pixels in the S1's azimuth IPR, the difference between the S1 and CPU IPRs is comparable to the variability between IPRs from repeated collects of the same scene using the same radar and acquisition parameters.

Computational Performance

Now we consider the amount of time the S1 takes to produce images. The S1 is a creation of a start-up and is not for sell on the open market. Sandia was unable to purchase actual hardware and had to do most development using software emulation. In the later stages of this work we were able to obtain access to an S1 system which we accessed remotely over the internet.

The system we had access to was a development system that ran at 50 MHz (as opposed to 200 MHz, typical of S1 systems). We gathered timing data for the S1 implementation on three different image sizes: 1024 x 1024, 512 x 512, 256 x 256. All experiments used the input 1024 x 1024 image's input data and parameters (pixel spacing etc.) with the exception of the number of output pixels. (The choice of 1024 x 1024 pixels for the largest image was driven by the fact that, due to memory limitations, 1024 x 1024 is the largest image our S1 CBP implementation can make on a 16 chip S1 system.)

While not strictly representative of actual SAR image formation practice, all three images used the same input, 2048 phase histories of length 3880 up-sampled by a factor of 6. The 512 x 512 and 256 x 256 images would have permitted processing every 2nd or 4th input phase history, respectively. The 512 x 512 and 256 x 256 sizes are peripheral to this investigation so it was decided that minimizing differences among the three image sizes (in this case, keeping the amount of input data constant) was more important than optimizing these cases.

Since the tile size is constant (16 x 16 pixels), the three different image sizes had different numbers of tiles. Varying the image sizes in pixels while holding the tile sizes constant enables characterizing the time-scaling of tile-related operations. Compute resources were scaled in direct proportion to the number of pixels (equivalently tiles). For the 256 x 256 image, one S1 chip was used. For the 512 x 512 image four S1 chips were used and sixteen S1 chips were used for the 1024 x 1024 image.

The following table records the time in milliseconds the S1 implementation of CBP spent in the various steps of the algorithm. The right three columns are for the three different image sizes on the S1. The size of primary interest is 1024 x 1204.

Image Size	256 x 256	512 x 512	1024 x 1024
Initialize	867	876	910
Process phase histories			
(1) Evaluate control points	3028	11268	43710
(2) Compute interpolation parameters	2883	11495	45899
(3) Distribute interpolation parameters	2458	9544	37853
(4) Perform interpolation	2200	2251	2401
(5) Distribute phase histories	10528	20680	41228
(6) Form image	748	746	750
Fetch Image	90	358	1434
Total	22829	57246	174213

Figure 7 Table of Execution Times

In contrast to the 174 seconds the S1 required to form a 1024 x 1024 image, Sandia's GPU implementation of CBP required 678 milliseconds on a Tegra X1 to form the same image using the same input data and parameters.

When interpreting the above numbers, the following facts should be kept in mind.
(1) The S1 system was clocked at 50 MHz rather than 200 MHz that most S1 systems utilize.
(2) Only modest effort was put into optimizing the S1 implementation of CBP.

With a 200 MHz S1 system and a more concerted optimization effort, it is not unreasonable to expect an order of magnitude improvement of the overall execution time.

Discussion

There are two main expenditures of time:
(1) performing the full-precision computation on the host processor on the sparse grid and
(2) transferring the data into the S1.

The time required by the purely compute portion, perform interpolation and form image, was almost independent of image size, approximately 2300ms and 750ms respectively, for a total of approximately 3050ms. This is substantially longer than the 678ms total time a Tegra X1 requires to form an image (I/O and all) of the same size. One major factor for this is that there are some operations where the S1 just had to do more work to do the same job.

First, a large fraction of the compute time was spent on interpolation. About 1/3 of the interpolation calculations were "unproductive". Lacking the ability to multiply integers, the code had to use repeated addition to update the interpolation parameters from the initial row to the row with an APE's pixels. This could have been done in one step if the S1 had the ability to multiply integers. Not only were these calculations not producing answers but its speed was limited by the APEs whose pixels are on the last rows of the tile. Only after all APEs updated the interpolation parameters to the row appropriate for the APE could they start producing answers that get used.

Another operation where the S1 was at a disadvantage is computing sin/cos. Pentium processors have sin and cos instructions. The S1 implementation of CBP computes a (sin, cos) pair using 170 instructions. Computing sin/cos using S1 data types was investigated early in this investigation. An approach using approx as the input and output data types was found to have undesirable numeric properties for this application due to non-uniform representation error of the arguments. The CBP implementation used a different strategy in which the input is an integer and the output an approx. While this approach required more computation, it had uniform bounds on the representation error of the input argument.

Since the S1 code to compute a (sin, cos) pair has no loops the 170 instructions of this code execute in 170 clock cycles. The Pentium line of processors has an equivalent fsincos instruction. The latency in clock cycles of this instruction varies considerably across the diverse Pentium product line. The latency of the fsincos instruction for an Atom core is 281 clock cycles [7].

A 16-chip S1 system has very high *aggregate* I/O bandwidth for data transfers within the grid. This number is not properly applicable to the S1 implementation of CBP. In our code, each output pixel is formed independently of the rest; the code performs no APE-to-APE communication. All I/O involved the CU. When viewed from the perspective of the high APE-to-APE bandwidth, the I/O through the CU was a severely limiting I/O bottleneck. The time it took to get the input data through the CU dwarfed the compute time. While going through the CU, the data transfers were essentially serial.

Singular Computing's Best-Practices Critique

The principle architect behind Singular Computing's approach to reduced precision computing analyzed Sandia's implementation of convolutional back projection on the S1. The critique suggested there are opportunities to speed the code up by a factor of 10 to 20. Given that the S1 implementation takes 174 seconds and the GPU implementation on a Tegra X1 takes 678 milliseconds on our benchmark 1024 x 1024 image, a 20x speedup of the S1 implementation would shrink but not close the gap between the two.

The opportunities for performance improvements identified in the critique are:

- Optimize code generated by the compiler:
  - Use compiler flags to utilize the hardware floating point of the ARM processor.
- Take advantage of the S1's huge compute bandwidth:
  - Move the computation of the phase and mapping parameters from the ARM host processor to the APEs and perform the computations using multi-precision fixed-point arithmetic.
- Optimize I/O operations:
  - Use double buffering for transfers between Host and Control Unit so that I/O and compute operations take place concurrently.
  - Harness the huge APE-to-APE I/O bandwidth by basing the extraction of the final image from the grid on APE-to-APE data transfers.
  - Avoid alternating cuWrite instructions with any other CU instruction by unrolling sequences of cuWrite operations. As a corollary, use the auto-increment capability of cuWrite.
- Optimize choice of instructions:
  - Replace Nova loops and ApeIf constructs with low-level instruction sequences emitted by eApe() and eCU() calls.
  - Unroll CUFor loops.
  - Do not specify a longer than necessary propagation delay for cuRead instructions.

With one exception, resource limitations of the project precluded attempting to exploit the opportunities identified in Singular Computing's review. The exception was supplying the compiler with the proper flags so the computations on the ARM performed floating point computations in hardware rather than in software. This reduced the overall wall-clock time by about one third.

The floating-point computations on the ARM are those related to computing the parameters for interpolating the mapping of samples onto pixels and the phase shift to apply before summation. The critique suggested even greater performance speed-up is possible if these computations were moved

out of the ARM and into the S1. Doing so would sacrifice full-precision floating point; the computations would need to be performed using multi-precision fixed-point arithmetic. The resulting multi-precision fixed-point would be much more elaborate than the rudimentary multi-precision, fixed point that is in the current implementation. Rather than storing values using two words, at least three two-byte words would be necessary, possibly even four. In addition, the S1's lack of an integer multiply instruction would entail emulating integer multiplication and division using the instructions that are available which would be quite slow. Singular Computing estimated multiplication to require about 1000 clock cycles per operation. (In contrast, the S1 can perform integer addition in a single clock cycle). Singular Computing anticipated that, even with the many cycles required to perform the calculations in multi-precision fixed-point, the massive parallelism of the S1 architecture would provide a significant net speed-up.

In addition to huge compute bandwidth, the S1 has huge I/O bandwidth for data exchanges between adjacent APEs. The current code extracts the final image from the S1 one word at a time in serial I/O fashion. The S1 architecture forces serial I/O because it can only transfer one value at a time between an S1 and the Control Unit. However, the method implemented requires multiple operations to transfer just one word. A faster I/O pattern, which minimizes the operations per transfer, is to have the APEs move data in a "bucket brigade" fashion. Once the "bucket brigade" is set up each transfer requires fewer operations.

In the same vein of minimizing the number of operations per task, the critique suggested replacing some constructs expressed in the high-level Nova language with lower-level constructs. An example is replacing the usage of Nova's inefficient ApeIf construct with low-level instructions using eApe() calls. The Nova language is new and the version of the Nova compiler Sandia used generates inefficient, overly conservative instruction sequences for some high-level constructs. Loop unrolling and auto-increment are other classic methods of maximizing the fraction of instructions executed performing useful work.

Conclusions

While this work demonstrated the ability to form SAR images on the S1 within the constraints associated with the S1 architecture (approx data type, absence of integer multiply and divide, etc.), the S1 has resource limitations that make it poorly suited in its current instantiation for forming SAR images.

(1) The amount of memory available is a primary limitation. The largest image we can form on a system with 16 S1 chips is approximately 1024 x 1024 pixels.

(2) I/O bandwidth between the Control Unit and the APEs is another primary limitation. Our GPU implementation of CBP can form an image on a Tegra faster than the S1 implementation reads an already formed image out of an S1 (700ms vs 1400ms for a 1024 x 1024-pixel image made from 2048 phase histories).

The actual computations on the S1 are relatively fast. They account for a fraction of the overall execution time (1.8% for a 1024 x 1024 image). For a 1024 x 1024 image size, data transfers to/from the APEs took 25 times as long as the computation.

Writing software for the S1 requires managing low-level details: data transfers, memory layout and management, and expressing computation at the level of individual operations. Explicitly specifying all the low-level details bloats the software. The CPU version of the CBP is less than 200 lines long. Largely due to having to code at such a low-level, our S1 implementation uses over 2000 lines of code to do the same thing as the 200 lines of CPU code.

The authors perceive the concept of reduced precision in general and its realization by Singular Computing to have great potential and are worth monitoring as reduced precision computing matures. The authors perceive the main challenges of using the S1 to form SAR images are with the current instantiation rather than the architecture. The S1 system used in this work is a prototype; the S1's limited memory size and I/O bandwidth are an engineering problem. Singular Computing expects to be able to scale memory and I/O bandwidth by a factor of hundreds in subsequent generations.

Acknowledgements

Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

The investigation presented in this paper benefitted immeasurably from the network of people supporting the authors' activities. The authors would like to thank the members of their Technical Coordinating Group, Casey Clark (NSWC-Dahlgren), Tom Lewis (AFRL-Eglin), Martin Heimbeck (AMRDEC), and Brian Smith (AMRDEC), for their guidance. Joe Bates of Singular Computing provided invaluable assistance through his instruction on best practices for utilizing the S1 architecture. Finally, the leadership of Dale Dubbert, Derek West and Kurt Sorensen of Sandia Labs created an outstanding environment for conducting this investigation.

References

[1] M. D. Desai, W. K. Jenkins, "Convolution Backprojection Image Reconstruction for Spotlight Mode Synthetic Aperture Radar," IEEE Transactions on Image Processing, vol. 1, no. 4, pp. 505-517, October 1992.
[2] C. V. Jakowatz, Jr., D. E. Wahl, D. A. Yocky, "Beamforming as a Foundation for Spotlight-Mode SAR Image Formation by Backprojection," SAND2008-1033C, Sandia National Laboratories, 2008.
[3] J. Bates, "Practical Approximate Computing," presented at Neuro-Inspired Computational Elements Workshop, Berkley, California, March 7-9, 2016.
[4] Singular Computing LLC, "Singular S1 Overview," Revision 2, August 2015.
[5] Singular Computing LLC, "Software Examples for Singular S1," Revision 1, August 2015.
[6] W. Kahan, "Further Remarks on Reducing Truncation Errors," Communications of the ACM, vol. 8, no. 1, pp. 40, January 1965.
[7] Intel, "Intel® 64 and IA-32 Architectures Optimization Reference Manual," pp. 16-25, December 2017.